亿级用户个性化品类推荐实战

前言

推荐是解决信息过载和挖掘用户潜在需求的技术手段。在美团 App 中，首页频道选择区承载了大量用户的消费需求，本文介绍了在亿级用户的场景下如何进行用户级的个性化推荐和品类推荐。

金刚区

金刚区是美团 App 首页的频道选择区（下图），金刚区有以下几个特点：

- 位置十分重要，是美团 App 的最主要的流量入口。
- 资源位较少，竞价规则和新业务扶持有冲突。
- 偏运营，顺序对业务的影响较大，且位置决定了顺序无法频繁变更，因为金刚区的用户习惯十分重要，我们曾经尝试改变美食和外卖的位置，导致美食的 CTR 下降。

中国联通 4G 11:34 100%

14° 北京 赛百味(外卖满30减9)

品牌日 美食

超火爆 电影/演出

酒店住宿

休闲娱乐

外卖

打车

KTV

赏花 周边游

火锅

生活服务

汽车服务

学习培训

高端酒店

超市/生鲜

全部分类

狠优惠

德来聚驴肉火烧（凉皮望...

外卖

25减18 ›

有格调

OOPS!西餐厅

西餐

金刚区排序规则

金刚区之前的排序规则是根据品类的转化率进行排序，这种排序规则会导致一些问题：

- 流量分发不合理。
- 排序结果没有用户维度的参考价值。

金刚推荐面临的问题

金刚区推荐与其他推荐的区别：

- 推荐的内容是品类，而不是 Item，这就意味着召回集较少。
- 可获取的数据都是用户针对 Item 的，而不是针对某品类，品类和 Item 之间存在多对多的映射关系，这一层转换是有损的。

金刚推荐三部曲

点击预估工作大部分时间都是在做特征挖掘和模型优化。仔细看各类推荐的文章、分享，使用的特征都大同小异，在实际业务需求中，我们究竟应该用什么样的特征才是最有效的。特征挖掘是一门艺术，需要对业务十分熟悉，还需要一点点的灵感。

> “数据和特征决定了机器学习算法的上限，而模型和算法只是不断逼近这个上限而已。”

我们在做金刚区推荐的过程中主要分了三个阶段：

- 第一阶段，用户行为推荐。
- 第二阶段，用户行为&用户画像推荐。
- 第三阶段，场景化推荐。

用户行为推荐

开始做金刚区推荐的第一件事情就是调研数据，我们花费了较多时间进行数据调研，实际中我们拿到的数据大多是根据埋点数据清洗后所得的数据，汇总了不同数据源的各种数据，调研数据的可用性以及整体分布情况，在调研数据的过程中，我们人工选择了一部分可用性较强的数据，并以此为基础完成了金刚区的第一版个性化。

这一版推荐主要使用了用户的历史行为进行数据统计，金刚区的推荐基础模型就是统计模型。

为什么会使用统计模型？

金刚区展出的品类中，可以被绝大部分人看到的品类所占比例较低，我们列举一种情况，比如，冬天的时候类似温泉这样的品类会是我们的热门品类，那么高意向和高点击会让温泉排名比较靠前，但是对于有些用户而言，我只对吃的感兴趣，我对温泉不感兴趣，这种情况下，温泉这个位置的流量在这部分用户下是无效的。显而易见的，用户曾经有过行为的品类即使不是他最感兴趣的，也是他大概率可能会有点击行为以及意向的品类。我们最初的想法就是，你喜欢什么，我们就给你展示什么。做法就是统计用户的行为数据根据权重计算后再归一化进行排序。

统计周期

- 一周、两周、四周。

使用的特征

- 用户点击/浏览。
- 用户下单。
- 用户购买。
- 用户收藏。

计算规则

Sort(Sigmoid(∑Count*Weight))

统计模型虽然看上去简单，但是针对我们的业务场景来讲，非常的有效，最简单的模型往往最有效，但是统计模型有一个弊端，就是当且仅当用户在统计周期内有行为数据时，才会有推荐结果，这一点不难理解，第一版统计模型上线后，覆盖率大概为50%，那么如何提高覆盖率就成了优化统计模型的重点。

针对覆盖率的提高，我们有以下改进方案，针对数据进行扩展：

- 设备 ID 到用户 ID 转换，扩展未登录场景。
- 扩大时间周期，覆盖更多用户。
- 扩展行为数据，增加收藏等数据。
- 品类层级关系转换。

用户行为 & 用户画像推荐

上面我们说到，统计模型效果固然很好，但是有很多模型本身的限制，一方面是无法做到100%的覆盖率，且推荐结果有限，另一方面是无法根据相关性进行推荐，一个新的用户或者用户行为较少的用户是无法获得有效推荐的。所以我们个性化推荐的第二阶段使用了用户行为&用户画像进行推荐。

特征工程

EDA，探索性数据分析，将我们的数据集可视化获取数据分布，比较变量分布情况，对数据进行检查等。

类别特征，针对类别特征我们可以选取不同的编码方式，如 One-hot 编码，哈希编码等，编码方式决定了特征空间的大小，编码后可以进行类别特征交叉组合。

数值特征，针对数值特征，我们需要根据第一步数据分析的结果，针对缺失值和离群值进行处理，对于缺失较多的特征需要考虑是否选取作为我们的可用特征，如果选取的话，那么缺失值是当做0值处理还是选择一个模型进行预测填充等等。并且针对数值特征，根据数值的范围，可以采取一定的处理，如范围较大的可以取对数进行缩放，如范围较小没有很好区分度的可以取平方进行拉伸，或者进行分箱转化为类别特征，如年龄等。

特征选择，通常可选的特征都非常多，我们可以选取部分特征进行训练，目的是简化模型，防止过拟合。

目前我们选用的特征有：

- 基础特征
    - 用户行为特征，浏览列表，购买列表，收藏列表.....

    - 用户属性特征，年龄，性别，城市……
    - ID 特征，品类 ID，BG，BU 及 one-hot 编码……
- 统计特征
    - 品类历史 CTR、意向率的均值等统计。
    - 品类历史下单率、购买率的均值等统计。
    - 用户历史的购买率，下单金额的均值等统计。

样本选择

过滤异常样本

过滤重复上报及上报有误数据，设置阈值，超过某阈值的数据删掉等。

抽样

我们采取的抽样策略是，正样本取全集，负样本按比例抽样，控制正负比例，一般情况下来讲正负例的比例应该控制在1：10之内，实际的比例应该根据业务需求来确定。

还可以采取分层抽样的方法，分层抽样可以根据样本中各类别的比例抽取，保证数据分布不会因为抽样而改变。

样本权重

为了平衡抽样带来的影响，我们需要在训练的时候为负样本增加权重。

模型训练

在模型选择的时候，我们尝试了 LR+FM 的组合模型，这里主要介绍一下 FM 模型，以及我们选择组合模型的考虑。

在很多情况下样本数据经过 One-Hot 编码后，大部分样本数据特征是比较稀疏的。通过观察大量样本数据我们发现，某些特征经过关联之后，与 label 之间的相关性会提高。表示特征之间的关联，最直接的方法是构造组合特征。样本中特征之间的关联信息在 One-Hot 编码和浅层学习模型（如 LR、SVM）是做不到的，目前常用的两种得到组合特征的手段是：

- 人工特征工程（数据分析 + 人工构造）。
- 通过模型做组合特征的学习（FM/FFM、深度学习等）。

事实上决策树也可以用作特征组合，决策树可以很方便的对特征做高阶组合，一颗决策树的每个叶子节点其实都对应一条特征规则。Facebook 就使用 GBDT 学习特征的自动组合。但是决策树和二项式模型有一个共同的问题，就是无法学习到数据中不存在的模式。例如，如果某种特征组合的数据不存在或者很少是无法做训练的，如果数据不是高度稀疏的情况下，基本特征组合都可以找到足够的样本，但是数据高度稀疏，二项组合就足以让大多数模式找不到样本训练，更高阶组合就更不必说了。所以，引入 FM/FFM 做特征组合是非常有必要的。

FM 的提出旨在解决稀疏数据下的特征组合问题。优化点在于通过奇异值分解将矩阵做了降维，复杂度从 O(n2) 降低到了 kO(n) 级别，使得单个特征不为0即可求得组合特征权重。k 就是 FM 中隐向量的维数，同时反映了 FM 模型的表达能力。

而 LR 在组合模型中起到偏移的作用，即新用户或者行为较少的用户依然可以根据品类的历史统计数据等协同得到较为热门的品类推荐。

在实际工程中，还可以将 XGBoost 用于特征选择。

组合模型使用同一个 Loss Function，针对 FM 模型特性，所以不需要引入 L1 正则。

场景化推荐

上下文特征

天气，变量，经纬度，距离，温度等。

实时流数据

分时段数据统计。

由于目前金刚区目前场景化推荐即将实验中，这里不做赘述。

数据太多怎么办？

在做金刚推荐的过程中，我们使用 Spark 进行数据处理工作，但是由于整体数据集过大，用 Spark 的过程中，也遇到了许多坑，由于 Spark 不是本文重点，所以这里不展开讲解，就分享一些问题的解决方案。

Spark 常见的问题有以下几个：

- 增加资源后没有太大改观，执行依然缓慢。
- 卡在某个 stage。
- Stage x failed 4 times, most recent failure。

针对以上问题，给大家一些建议。

- 增加资源后一定程度上会减少作业的执行时间，但是盲目增加资源不是有效的，甚至有时候是无效的。我们需要合理的配置 Spark 的参数，以优化我们的作业，比如：
    - spark.sql.shuffle.partitions，执行 SQL 中的 shuffle 操作中默认分区数量，比如 group by、join 等，默认为 200，事实上 200 个分区数量在大部分场景下都不够，适当调整这个参数对于整个 job 优化非常有效。
    - spark.default.parallelism，每个 stage 默认 task 数量，Spark 中 Task 被执行的并发度 = Executor数目 * 每个Executor核数，如果 Executor 的数量较多，但是单个 stage 的 task 数量不够，就可能导致只有 1~2 个 Executor 在执行，其他 Executor 在闲置，这样就会造成资源浪费。
- 卡在某个 stage。
    - 这种情况一般是因为数据倾斜，建议是首先观察一下耗时最久的 stage 是第几个 stage，然后观察自己的代码不管是 SQL 还是 RDD 操作，根据 shuffle 操作进行一下划分，确定一下耗时最久的 stage 出现的位置，然后将数据集 countByKey 输出一下，查看一下数据集分布情况，然后有几种解决方案：
    - 根据多个 key 组合后再重新分区，这样会有效减少单个 partition 中的数据量。
    - 用 key+随机数作为新的 key，这样会避免单个 key 数据太大的情况，但是如果单个 key 数据很大在后续 shuffle 操作中依然会很消耗性能，如果后续只有 map 操作是可以这样来避免倾斜的。
    - 根据倾斜情况，进行数据预处理。
- 第三种情况有很多种可能，直观上来看是说某个 stage 的重试次数超过了4次导致失败，实际遇到这种情况应该在执行的时候查看 failed 的 stage 中的 error log，查看具体失败原因，导致这种原因有以下几个可能。
    - join 等操作导致 Executor OOM
    - join 操作会导致 shuffle 操作，会将两个数据集聚合在同一个 Executor 上然后进行 join 操作，如果两个数据集比较大可能就会导致 Executor OOM。报错如下，“org.apache.spark.SparkException: Job aborted due to stage failure: Task 1678 in stage 9.0 failed 4 times, most recent failure”，虽然表面看上去报错是因为某个 Executor 超时并且重试失败，但是在作业进行的过程中观察 stage 失败原因会发现是由于 Executor OOM 导致的。这时候可以用 broadcast 代替 join 操作。具体步骤如下：

（1）选择较小的数据集进行广播。

（2） 遍历较大的数据集进行 map 或者 mapToPair 操作，两个数据集的 key 相同则进行 join 操作。

- cartesian 等操作导致 driver OOM。

- 由于笛卡尔积是将结果集拉到 driver 上做聚合，所以如果结果集很大的话就会导致 driver OOM，这个问题的解决方案和解决 Executor 的方案差不多，同样是将小表广播，大表进行遍历，拆分的时候需要使用 flatMap 等

Spark 是非常灵活的，在使用中需要对宽窄依赖，各操作的具体执行方法等有一定了解，尽量避免 shuffle 操作，合理配置参数，即使是数以百亿计的几十 T 数据，同样可以在一小时内完成操作。

总结

本文结合美团 App 本身以及金刚区的业务特点进行推荐的一些经验和过程，目前部分策略已经上线，并且取得了正向效果。希望可以给即将做推荐、正在做推荐或者对推荐感兴趣的工程师同学带来一些启发和思考，最后，希望以后可以为大家带来更多有价值的分享。

> 转载自 https://gitbook.cn/gitchat/activity/5ac465d1817f7f29e510aaad